volkszaehler.org

the open smartmetering platform

Zusammenfassung

volkszaehler.org ist ein freier Smart Meter im Selbstbau. Die anfallenden Stromprofile bleiben dabei unter der Kontrolle des Nutzers.

Seit dem 1.1.2010 müssen Stromversorger ihren Kunden für Neubauten so genannte "intelligente Stromzähler" (Smart Meter) anbieten. Der Kunde soll dadurch seinen Stromverbrauch analysieren und optimieren können.

Das dabei anfallende Stromverbrauchsprofil erlaubt einen sehr detallierten Einblick in den Tagesablauf des Nutzers (wann steht er auf? wann geht er in's Bett? wann kocht er? wie oft verwendet er seine Spülmaschine? verändert sich sein Verhalten? ...).

Darum sollten diese Daten ausschließlich für den Nutzer selbst zur Verfügung stehen - und das geht nur, wenn man sich den Smart Meter selbst baut. Mit einem Materialeinsatz von ca. 100 €, etwas Geschick und Zeit lässt sich das mit Hilfe eines Standard-µC-Moduls aufbauen.

Inhaltsverzeichnis

1 Einführung 3
2 Aufbau 3
2.1 Controller 3
2.2 Frontend(s) 4
2.3 Backend 4
3 API 5
3.1 Gruppierung 5
4 Implementierung 5
4.1 MVC 5
4.1.1 Router 6
4.1.2 Kontext (Controller) 6
4.1.3 Datenabstraktion (Model) 6
4.1.4 Ausgabe (View) 6

1 Einführung

In diesem Artikel werden wir das Projekt vorstellen, einen generellen Überblick über die einzellnen Kompontenten geben und auf die Implementierung eingehen. Ich möchte damit den Einstieg in das Projekt erleichtern und neue Entwickler motivieren sich zu beteiligen.

Einzellheiten zur Installation/Konfiguration eines eigenen volkszaehler.org Setups finden Sie unsere Mailingliste[1] und im Wiki[2].

Neben dem **Schutz der Privatsphäre**, hat das Projekt auch noch einige andere Ziele:

- eine kostenfreie Alternative gegenüber den kommerziellen Messstellenbetreibern anbieten
- dem Nutzer ein Bewusstsein über seinen Verbrauch/Nutzungsverhalten aufzeigen
- Energie intelligenter zu nutzen
- die Breite Masse erreichen[3]
- offene Protokolle und Standards zu vorrantreiben

Wir haben es uns zur Aufgabe gemacht diese Ziele durch eine lückenlose Kette von quelloffener Soft- und Hardware zu erreichen. Der Nutzer soll die Möglichkeit haben jeden einzellnen Schritt nachvollziehen zu können. Daraus folgt nicht, dass wir jede Zeile Code selbst geschrieben haben. Wir nutzen eine Reihe anderer Software, die aber wiederum selbst quelloffen ist.

2 Aufbau

Das Projekt lässt sich in drei Bereiche aufteilen, die untereinander über eine spezifizierte API kommunizieren. Diese drei Module grenzen sich lokal, durch die verwendeten Technologien und ihre Aufgabe voneinder ab. Alle Module sollen untereinander austauschbar sein und sind in mehreren Varianten verfügbar. So kann den individuellen Bedürfnissen nach ein angepasstes Setup aufgebaut werden.

2.1 Controller

Die Aufgabe der Controller ist es Sensoren auszulesen und diese Daten direkt an das Backend zu senden. Meist sind sie direkt mit den Zählern/Sensoren verbunden. Ein typischer Ort wäre also der Sicherungskasten. Diese Controller sind dann meist in das lokale IP Netzwerk eingebunden und senden ihre Daten

[1] volkszaehler-dev@lists.volkszaehler.org
[2] http://wiki.volkszaehler.org
[3] Daher stammt auch der Name **volks**zaehler.org

an ein Backend. Ausgehend von Prototyp sind im volkszaehler.org Projekt bisher die Atmel AVR/Ethersex basierenden Controller am meisten verbreitet.

Als Sensoren werden hier für das Erfassen des Verbrauchs Impulszähler eingesetzt. Diese signalisieren dem Controller durch einen elektrischen Impuls, das eine bestimmte Menge an elektrischer Energie, Wasser oder Gas verbraucht wurde. Auch skalare Messgrößen wie z.B. Temperatur, Luftdruck oder Luftfeuchtigkeit sind möglich, werden dann aber nicht mehr durch Impulse erfasst.

Ein typisches Setup für ein Einfamilienhaus könnte folgendermaßen aussehen:

- 3x 1-phasige S0-Stromzähler für den Hausanschluss[4]
- 1x S0-Wasserzähler
- 1x S0-Gaszähler
- 1x Sensor für die Außentemperatur

2.2 Frontend(s)

Die Frontends visualisieren Messwerte und können zur Verwaltung genutzt werden. Durch ein zentrales Backend haben wir die Möglichkeit die Messwerte an mehrere Frontends gleichzeitig zu verteilen. Verschiedene Frontends, können die Daten dann unterschiedlich darzustellen.

Das Web-basierte Frontend für den Browser ist das bisher einzige Frontend und ist quasi eine Referenzimplementierung der API. Es unterstützt die volle Funktionalität des Backend und kann auch zur Verwaltung von Zählern genutzt werden. Es nutzt AJAX um Daten dynamisch nachzuladen.

Inspiriert vom Wattson[5] haben wir uns entschieden eine ähnliche interaktive Variante dieses Moodlights zu entwickeln. Derzeit experimentieren wir mit fnordlichtern[6].

Die möglichen Visualisierungsmethoden sind praktisch endlos. Damit auch die Anzahl der möglichen Frontends. Frontends dürfen nicht nur als abgeschlossenes System betrachtet werden. Denkbar sind auch Plugins in bestehende Systeme (Social Networks, iGoogle, Twitter).

2.3 Backend

Im Backend werden die Messwerte der Sensoren gespeichert und ausgewertet. Es handelt sich hierbei um ein PHP Skript welches auf einem Webserver läuft und somit die Schnittstelle zwischen Controller und Frontend ist. Daraus folgend muss es von den Controllern sowie von den Frontends via HTTP erreichbar sein. Ob der Backendserver nur über das lokale Netzwerk oder auch über das Internet erreichbar ist, wirkt sich nur auf die Erreichbarkeit durch die Frontends und die Controller aus. So kann es durchaus erwünscht sein die Daten nur im lokalen Netzwerk vorzuhalten um sie gegen Zugriff über das Internet zu schützen.

[4] jede Phase erhällt einen Zähler
[5] http://www.diykyoto.com/uk/wattson/about
[6] http://wiki.lochraster.org/wiki/Fnordlichtmini

3 API

Die API definiert eine Schnittstelle zwischen Controllern und Backend, sowie zwischen Backend und Frontends. Die API orientiert sich am REST Entwurfsmuster. Dabei kann jeder Zähler/Sensor/Gruppe (im folgenden als "Entity" bezeichnet) durch eine weltweit eindeutige UUID referenziert werden. Diese UUID werden entsprechend RFC 4122 von Backend generiert und vergeben. Dieser 128 Bit lange Identifier sichert uns auch gleichzeitig die Privatsphäre. Die UUID ist praktisch eine Zugangskennung und sollte immer vertraulich behandelt werden. Da wir keinerlei nutzerbezogene Daten speichern kann diese UUID nur durch den Zugriff eines Nutzers zugeordnet werden. Daher empfiehlt sich der Einsatz von verschlüsseltem HTTPS.

Als Ausgabeformat nutzen wir das leichtgewichtige und menschenlesbare JSON.

Eine Referenz unserer API befindet sich im Wiki.

3.1 Gruppierung

Zähler und Sensoren können zusammengefasst werden. Diese Aggregatoren besitzen selbst wieder eine UUID und können dadurch beliebig tief verschachtelt werden. Hirachische Strukturen wie z.B. Mehrfamilienhäuser, Wohngemeinschaften, Hotels, Gemeinden & Städte können dadurch nachgebildet und gemeinsam ausgwertet werden.

4 Implementierung

Dieser Abschnitt konzentriert sich auf die Implementierung des Backends. Das Backend ist in PHP programmiert und kann auf jedem LAMP-ähnlichen System betrieben werden.

Der Code ist stark Objekt orientiert. Daher müssen wir PHP 5.3 vorraussetzen. Diese Objekt orientierte Programmierung erlaubt es uns den Code übersichtlicher und modularer aufzubauen. Zukünftige Erweiterungen, neue Zählertypen können dadurch leichter in das System integriert werden.

4.1 MVC

Das Backend wurde nach dem **M**odel **V**iew **C**ontroller Entwurfsmuster aufgebaut. Jeder Request an das Backend muss über die Datei `/htdocs/backend.php` erfolgen. Sie ist quasi der "Haupteingang"[7].

Dort wird zunächst der PHP Interpreter konfiguriert (automatisches Laden von Klassen, der Konfiguration, Fehlerbehandlung, Aufbau der Datenbankverbindung). Danach wird direkt der Router initialisiert.

[7] Dadurch ist sie auch das einzige PHP Skript, das über das öffentliche "htdocs" Verzeichnis zugänglich sein muss

4.1.1 Router

Der Router ist das Herz der MVC Struktur. Er leitet den Request an den zuständigen Kontext[8] weiter und antwortet mit dem korrekten Ausgabeformat.

Alle Anfragen an das Backend müssen einem bestimmten Schema entsprechen. Hier wird auch der zuvor angesprochene Einsatz von REST deutlich.

```
http://server:port/path/to/volkszaehler/backend.php/kontext/uuid.format?paramters=values
```

4.1.2 Kontext (Controller)

Das Backend hat verschiedene Aufgaben zu bewältigen. Für jede dieser Aufgaben gibt es einen eigenen Kontext, der praktisch die ganze Logik enthällt:

- Daten erfassen/ausgeben
- Entities erstellen/bearbeiten/abfragen
- Eigenschaften des Backends abfragen

4.1.3 Datenabstraktion (Model)

Zur Datenbankabstraktion setzen wir Doctrine 2 ein. Dieses Database Abstraction Layer (DAL) erlaubt es uns datenbankspezifische Eigenheiten, SQL-Dialekte ähnliches zu ignorieren. Doctrine unterstützt verschiede Datenbanken:

- MySQL
- SQLite
- PostgreSQL
- Oracle

Aufbauend auf das DAL kommt der Object Relational Mapper (ORM) von Doctrine zum Einsatz. Wir können mit den Daten nun als Instanzen von PHP-Objekten arbeiten. SQL Queries sind nicht mehr nötig. Für weitere Infos empfehle ich die Dokumentation des Doctrine Projekts.

4.1.4 Ausgabe (View)

Das Backend kann in verschiedenen Formaten antworten:

- JSON (Standard)
- XML
- CSV

[8] Wir nutzen hier den Begriff “Kontext” um eine Verwechslung mit dem Hardware-Controllern zu vermeiden. Entsprechend des MVC-Patterns ist hier der “Controller” gemeint.

- Klartext
- Jpeg, PNG, Gif

Dabei ist JSON das bevorzugte Format, das auch von dem Webinterface genutzt wird. Die anderen Formate sind als Alternativ für Endgeräte ohne JSON Unterstützung gedacht. Wir können nicht garantieren, dass diese Formate den gleichen Funktionsumfang wie das JSON Format besitzen.